SICS/T-89/8914

On the Usage of Knowledge Based Techniques in Configuring Computer Systems: A Case Study

by

Jerker Andersson and Per Andersson

SICS technical report
T89014
ISSN 1100-3154

RIT/SICS/ABB
Jerker Andersson
Per Andersson

1 Introduction

To begin with we would like to discuss different approaches to the design of a knowledge based system (KBS) in general.

1.1 Analysis and Synthesis

The roles possible to assign to a KBS can be divided into two groups that are clearly distinct. Either the system performs a synthesis or else the task is an analysis.

1.1.1 Some Background Aspects

A compound KBS performing both an analysis and a synthesis would then necessarily be regarded as consisting of at least two systems. Either they are in sequence and are hence independent or they exchange information, being disjoint agents.

Most KBSs existing today belong to the class of Heuristic Classification. This is a subspace of the analysis problem domain where some pre-enumerated solutions are considered and the solution, that best agrees with the observed data is chosen. This kind of system is what you could expect to realize with a limited number of rules in a commercially available tool.

In contrast to analysis there are no (simple) general tools available to solve simple synthesis problems. This is probably due to the fact that the simple general problem is yet to be discovered. Regarding synthesis problems, the search space swells rapidly as the number of rules are increased, as compared to heuristic classification, where more rules tend to narrow the search path.

RIT/SICS/ABB
Jerker Andersson
Per Andersson

1.1.2 Types of Analysis

The actual span of analysis concerns everything that can be formalized and interpreted to yield a conclusion. An interpretation can be made in order to control a system, simulate it, i.e. predict its behavior, or make an identification. Identifying can be done either in order to verify a system or else to diagnose it and establish probable faults.

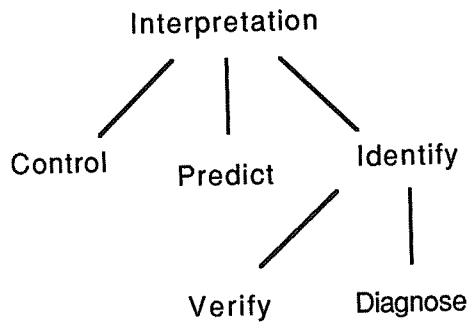

Fig 1. Analysis type of problems

An example of a more skilled general tool is EMYCIN that have developed from a specific application to become a KBS shell. All domain knowledge have been removed and only the inference engine and data structures remain. The system designer will only have to format his data according to EMYCINs guidelines in order to have a working application.

1.1.3 Types of Synthesis

The synthesis problem domain, as well as the analysis problem domain, can be divided into several subdomains. A synthesis, when viewed as a process of construction, can be any one of specification, design or assembly. A design problem could be regarded as either a configuration or else a planning problem. The configuration concerns the structure of the construction and planning establishes a process to use in the assembly, both of these are guided by a specification. These are constrained by the specification. When the structure and the process are both known assembly of the product takes place.

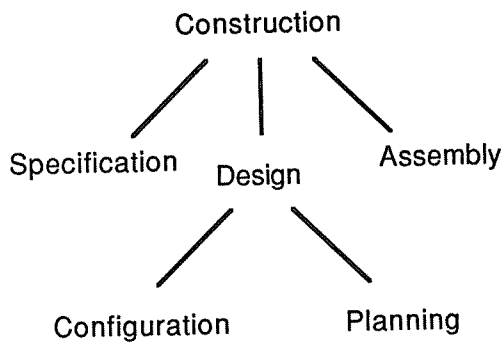

Fig 2. Synthesis type of problems

RIT/SICS/ABB
Jerker Andersson
Per Andersson

A well known assembly system is R1 [Bachant & Dermott 84]. The R1 project was initiated in 1978 and has developed over the years to become a full-fledged expert with over 3300 rules and a database of some 5500 components. R1, given an order for a computer system, determines what substitutions and additions are needed to make it consistent and complete. The result is a modified parts list as well as plans showing the spatial and logical relations of the components, usually there is some 50 to 150 of those. R1 knows far from everything about the systems it designs but still works at a far deeper level of detail than any human technical editor.

1.2 Some Methods to Compute Solutions

Below follows a short narration of some methods selected from suggestions made by [Clancey & Rothenberg 87]. These methods represent contrasting approaches to the diverse types problems presented above.

1.2.1 Heuristic Classification

As stated earlier this method is a selection from pre-enumerated solutions. A well-defined decision tree leads the inquiring person straight down the homeward path. Case specific information such as test data can be provided when a rule demands it. A simple "if-then-else" approach suffices.

A useful expansion of the rule evaluation in any method would be reasoning with uncertainty. Assign a probability to each possible choice at every decision point stating the chances of that choice being correct given the premises. The values used can be based on statistics, test values or expert appreciation by the system expert or the user. Or they could be functions using probabilities computed for subproblems.

1.2.2 Search of Solution Space

By using a systematic application of solution operators the initial problem is transformed until it reaches a halt state. A halt state is recognized by the fulfillment of one or more criteria characterizing a solution. During the course of transformation a considerable amount of backtracking may occur depending on how well the operators are selected. Some kind of look ahead or heuristics improve performance, for instance by using partial evaluation or local search.

If presented with a proper set of start-up information this method would be well suited for synthesis. The problem lies in mimicking the technical editor in his work. Exactly why do we take certain steps while constructing, say, a football tactic? Also, the representation of something complex to be dynamically put together often calls for great domain knowledge and ingenuity.

1.2.3 Blackboard Model

This method has been described as several person laying a jigsaw puzzle without communicating with each other. Instead they only observe what is happening on the "blackboard" and when someone notices

that one of his pieces will fit in, he puts it up on the blackboard. Hopefully someone else will now discover that one of his pieces will fit in with the new one, the solution thus developing.

This is an opportunistic incremental generation of a solution. Obviously we could use this method for synthesis as well as analysis. The most appealing advantage is the explicit separation of different tasks or expert roles. A drawback could be the problem of realizing when a piece is missing or should be removed.

1.2.4 Derivation and deduction

This is a classical science, concerned with establishing a proof that a given statement is obtainable or correct given the premises. Facts and rules are present in a knowledge base and the user supplies a goal that he wishes to establish. The goal is split into several subgoals using an appropriately selected rule. The subgoals in their turn will be handled in the same way. This is done recursively until so called ground facts are obtained; the goal is proven. This method of resolving is called backward chaining.

Forward chaining would then be an attempt to combine facts so as to prove a rule. A rule thus proved could be asserted as a known fact, thereby broadening the number of ground facts available. Forward chaining would be a tremendously inefficient way to prove a given goal since we would not know which rules there would be in a correct proof chain. All rules would have to be considered. But it is often used initially, to expand the number of ground facts making backward chaining faster.

1.2.5 Debugging

By this is meant checking a solution for inconsistency and incorrectness trying to deduce the source of errors found. This method could also be applied to "optimization" bugs, pointing out constructs in a system that are inefficient.

Naturally debugging could only take place if there exists a suggestion for a solution. If a solution is provided there would be a rule base describing all permissible combinations of facts yielding independent parts of a solution. If rules could not be found for some part of the solution that would imply an error. Another approach would be a list of inconsistency rules naming all instances of combinations not allowed. If any one of these rules could be proved to hold then there would be a fault at hand.

1.3 Some Process Models

All expert systems are based on some model of reality and as was previously stated most successful systems are those based on classification models. The following discussion concerns different types of models as proposed in [Clancey & Rothenberg 87]. We will later refer to this section when defining our model used for configuring, both general and specific.

1.3.1 Classification or Simulation

Classification implies the recognition of patterns of features characterizing objects, events and processes. Patterns might be incomplete or indetermined yielding different matchings or requiring completion by other means. The target domain must be well defined in some aspect so as to establish criteria for completion or failure.

Simulation on the other hand, does not need to terminate, other than for a practical reason; we would like to observe what has happened with our system. In order to simulate, we use cause-effect descriptions describing our objects, events and processes. Starting from some initial setting we then apply these "rules" to the objects causing events to occur within processes and processes to communicate. A typical simulation would have to utilize parallel, or pseudo-parallel, execution.

1.3.2 Behavioral or Functional

The cause-effect operations present in a system can be modelled in two different ways. Most expert knowledge in the real world is stated in a behavioral form; "when I do this that happens, and when I do this something else happens". The representation of such data would be on the form of state descriptions and transitions. The underlying functions would be hidden from view.

The knowledge engineer could use these associations as they are found, or they could be scrutinized and analyzed to find the structural and functional composition yielding the behavior. This could of course be difficult or even practically impossible. However if there are functions to be found, the solution would be generalized and hence the system more "knowledgeable". Also the functional approach lies closer to the classic algorithmic computer program.

1.3.3 Flat or Hierarchical

Without any abstract structural modules or functional operators the system would be considered flat. If computations are local this might improve on understanding and flexibility since all data and relations are available on the "surface". Intuitively the flat structure is still partitioned into different regions much like a map can be colored with different aspects of political or socio-economic nature.

On the other hand, a complex problem would be hard to handle if it is modelled without abstraction and modularization. The flat structure of complex relations would not provide an overview. If instead modules are identified and relations can be handled on different levels then a hierarchical composition of primitives into abstract modules and operators should be used. Local data will become local and the focus will be on relevant relations for the subproblem being studied.

RIT/SICS/ABB
Jerker Andersson
Per Andersson

2 Observations Regarding Configuring

When we consider a product that is subject to configuration we may do so on several levels of abstraction. Trying to differentiate between these can be the main obstacle when establishing what methodology to employ in solving the problem. This implies that we must be careful when deciding what we would like to configure, otherwise problem complexity might become overwhelming or the need for highly detailed expert skill to cumbersome.

2.1 Complexity of Configurations

As hinted, it is of great importance to establish as clear a view as possible of the conceptual model needed for modelling a system. A configuration could be viewed as a relational model based on objects and constrained by rules delimiting the target domain. Such a view, even with few rules, provides an enormous amount of possible solutions and necessitates exhaustive heuristic selection criteria. At the other extreme there might be a completely operational model trying stepwise to emulate the decision made by the expert. We would then need a great number of rules to appreciate different aspects for every possible scenario.

The set of relevant parameters modelling a system might be quite dissimilar depending on what aspects are considered essential. Since each additional parameter rapidly swells the solution space it would be advantageous to keep these at a minimum. We could not hope to avoid the combinatorial explosion altogether, it lies at the heart of the configuration problem as such. Rather, it is the human way of dealing with the combinatorial effects in configuration that we would like to capture and model.

Another reason for minimizing the set of parameters would be a wish to avoid redundancy. If one particular circumstance can be modelled in several ways this might produce peculiar effects. Both in terms of "doubling the value" for that circumstance and the possibility of different modelling traditions evolving for essentially the same problem. This last point is made in order to highlight the chance for undue parallel development. It may not always be undesired, such development may sometimes be helpful in adding depth and understanding to a complex topic.

2.2 Acquisition of Skill

A human technical editor often uses many problem solving methods in parallel to tackle a problem. To identify these and extract the information to model them is known as knowledge acquisition. Then, in order for a computer to put this knowledge to use, we must have an efficient way of expressing facts and rules. Also there must exist some channels for backfeeding. In order to refine the expertise contained in an expert system one must be able to understand how an observed behavior is related to the system content and its structure. Preferably the user of a system, or the experts that provided the knowledge, should be able to update and edit the know-how. A system that could learn by experience would be nothing to bank on, induction not being one of the basic virtues of computers.

RIT/SICS/ABB
Jerker Andersson
Per Andersson

Rapid prototyping could serve as a starting point for designing a configuring type of system. Several approaches could thus be tried. However, in order to carry out any work that is not trivial, a fairly large amount of domain knowledge would probably have to be formalized. Moreover there should be non-trivial test examples displaying the same kind of quirks that a real case could be expected to show. It could even be claimed that any candidate plan of action should be tested for its capability of handling a good sized portion of rules and data at as early a stage as possible.

2.3 Abstraction

Configuring implies matching parts together to construct a new thing to fulfill a complex need. This composite item may then itself be part of a larger structure. Obviously we could use abstraction for improving understanding of the structures to be handled. It is not clear, however, whether or not there are hierarchical functions in our problem domain.

So, how do we benefit from abstraction? It is essential in the avoidance of combinatorial explosion to be able to disregard irrelevant information. When parts of a system have been readily packaged we want to handle the result as one object. The attributes of such a composition would be an aggregation of the attributes of the parts. Hopefully we could thus reduce the number of parameters that needs to be satisfied. One could imagine there being difficulties in aggregating attributes correctly and this could well be another task for the expert. However, in this case we will show that there is no such problem in our generalized simple model.

2.4 Suggestion for a General Solution Methodology

There are some observations that suggest that there might be a general structure extractable from the work we have done. It will be described in short.

We have found that our problem domain consists of two systems. One describes what is to be performed, this we have called "uses", and the other describes a system able to provide the services requested, we call this part "supplies". Both of these are abstract hierarchical structures with basically primitive functions as leaves. We believe that these primitives can be chosen so that they map exactly onto each other, i.e. there is a one to one correspondence between the leaves in the "uses" system and the "supplies" system. The reason to pick such a representation is to make an easy match from the demands to the services, thereby gaining in simplicity. The whole construct would then be a lattice, or a directed network flow diagram with a cut prescribed by the problem domain as the border of the two systems.

RIT/SICS/ABB
Jerker Andersson
Per Andersson

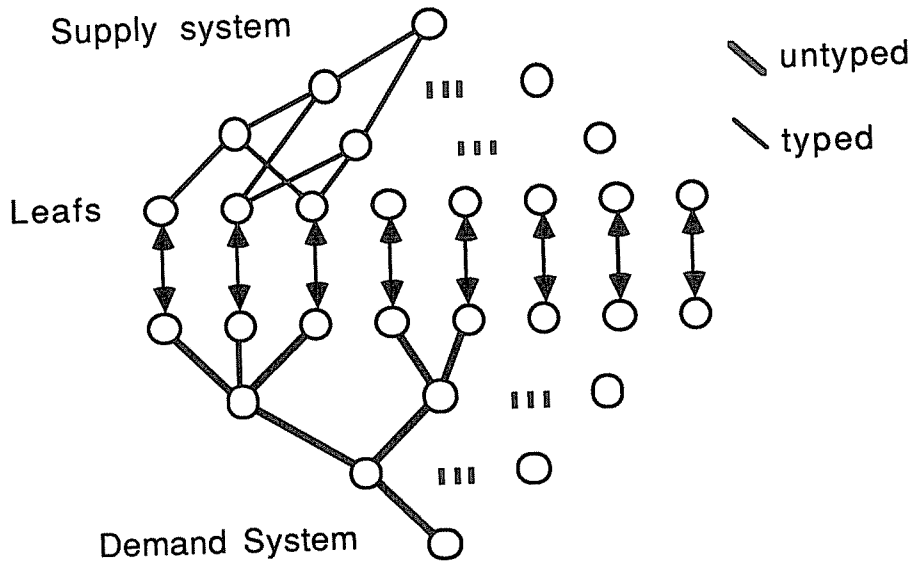

Fig 3. The systems viewed as a flow graph

The demand system can, for our purposes, be regarded as a flat structure, the hierarchy aiding the user in gaining an overview. The actual granules of "demands" to be configured are to be found in the leaves only.

The nodes and arches in the "supplies" graph will have to be typed. Each node provide its sons with a "supply" of predetermined type and put "demands" on its father(s). There may be multiple ancestors to provide a node with several resources that it may need.

However, to accomplish this the leaves may not be strictly primitive from the hardware or software point of view. One Basic function might well be composed of several such basic units as I/O and program modules. We therefore need to agree upon a set of such "low-level" abstract functions to use as our leaves.

3 Xces, An Example Control System Designer

3.1 Introduction

xces, Experimental Configuring Expert System, is a knowledge based tool intended to configure computer systems for process control. The design has been kept as general as possible and, to date, little of domain knowledge has been built into the abstract machine.

3.1.1 System

The development of *xces* has been done using ZYX prolog, by ZYX Sweden AB, on Macintosh SE and Macintosh II machines. The current version runs under ZYX 1.5 or 2.7 and preferably more than 1 Mbyte of memory to allow for significantly large data structures. It has been the intention of the designers to keep as close a watch on the Macintosh

User Interface Guidelines as possible in order to make a user friendly man-machine interface.

3.1.2 Windows

There are several types of windows in the interface. The main window is the Edit window, where the currently modelled process will be displayed. Other important information carriers are the Information window, Commentary window and System window. They provide information about the current process node as well as comments made by the allocation algorithm and proposed configurations. There is also a host of dialog boxes that prompt the user for information.

3.1.3 Menus

Menus provide commands. Similar commands or commands operating on the same type of objects are grouped together in the same menu. First there are File and Edit menus that are roughly the same as in any other Macintosh application. Then follows three menus for creating new structures. These are named Create, Macro and Block. Finally there is one menu for making marks in the process model and one for accessing the dictionary. The different commands be found in the users guide. Concepts such as "macro" and "mark" will be explained further later on in this paper.

3.2 The object domain

The object domain consists of two parts: a process to be controlled and a control system to control it. The former is to be modelled and the latter to be configured.

3.2.1 Conceptual model of a process

The process being modelled is recursively broken down into subprocesses until a lowest level of objects representing control functions is reached, these are called function blocks. Functions, e.g. pid-regulators, thermal readings and valve control, required to control a process are more or less dependent on each other. It is essential that we are able to capture this aspect in the modelling of a process. Hence a process, or subprocess, is a member of one of three classes: "same", "diff" and "none". Either all its subprocesses are to be kept together, or else they should be placed apart in different control stations. The third class is a "don't care"-class. One could argue that bindings of variable strength would be appropriate and we agree; however this possibility has not been included and the use of such a feature is not at all clear.

Moreover, in order to capture relations not imposed by functional dependencies, there exists a command to mark processes with different markings. For instance it could be useful to be able to separate different technologies e.g. high voltage into one cabinet and digital signals into another. Also, physical constraints other than those modelled functionally could be dealt with, such as machines situated in different buildings

3.2.2 Conceptual model of a control system

A control system consists of several computer, control stations, interconnected by a bus. A station may contain several types of hardware and software such as high voltage and logical I/O and program modules that compute values and control hardware. In this version we have greatly simplified the system. It consist of control stations, I/O-boards, I/O-functions and program modules corresponding to the different blocks mentioned above. Such things as power supply, busload and cabling have been merrily ignored.

Configuring should yield a number of stations supplied with boards. For each station there should be a list of functions and collectively allocated subprocesses that are supposed to run on that particular computer. The requirements of the items on the list should be provided by the station and its boards.

3.3 User model of *xces*

The conceptual model of the *xces* system has been designed to be used in an edit-run cycle. This means that we start off by modelling a process, then we let *xces* design a solution. This solution is then used as feedback by the user and it will induce changes in the model.

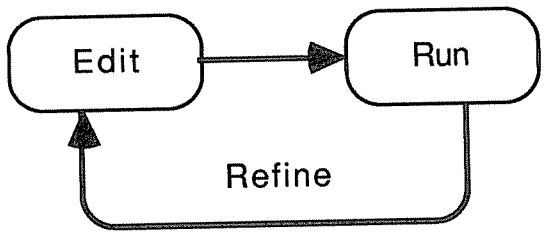

Fig 4. Edit-run cycle

The *xces* tool consists of four disjoint interacting parts: editor, translator, configurer and a lexicon. These have clearly defined interfaces both among themselves and towards the user. Below follows a brief description of the different modules.

3.3.1 The Editor

This is a semi-graphical editor displaying a hierarchical structure of a process model. All information contained in a node is local, there is no propagation or inheritance.

Each node contains information about the process or functions it represents. Whenever a node is selected all information about it will be displayed in the Information window.

To help in the design there are, besides cut and paste, several functions e.g. multiplication, macro definition and the possibility of custom designed function objects.

3.3.2 The Translator

The representation of a process used by the editor is equipped with several aggregate functions to provide a powerful development environment. These are converted into corresponding structures containing basic building blocks. Thus the whole structure is expanded to a "ground" format that is prescribed by the configurer.

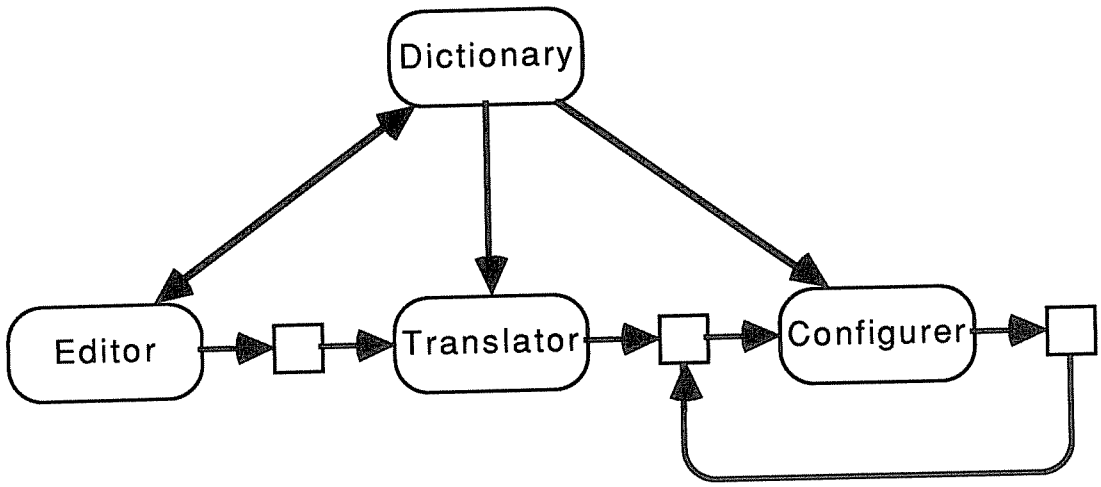

Fig 5. Interrelationship of *xces'* different parts.

3.3.3 The Configurer

Solves the problem of configuring a system once the desired functions of a process have been modelled. The allocation of functions to process stations and subsequent reservations of resources are achieved by a highly deterministic algorithm, detailed in section 3.5.3. The configurer can be viewed as a packaging algorithm, cramming as much as possible into each machine, e.g. the travel agency problem. However the distribution of functions among stations will be restricted and guided by the class type of each node.

3.3.4 The Dictionary

All domain specific knowledge and data are kept separate from the tool proper. In order to achieve a general and flexible tool, a clear definition of what is program and what is data must be thought out. Therefore we have a dictionary. An entry into the dictionary consists of a subject with several data items selectable by keywords. The dictionary is supposed to store anything that is needed by the tool and is not bothered by format or contents.

3.4 Inference engine

The inference engine of *xces* resides in the configurer module. It has unfortunately not been very stringently implemented or defined but the rules, data and mode of operation are laid out as follows:

RIT/SICS/ABB
Jerker Andersson
Per Andersson

3.4.1 Rules

It uses as rules the relationship between processes and functions as described by the process model. These are augmented by rules guiding the allocation of resources and rules to enforce node type restrictions.

It is unfortunate that rules regarding process and function relations are implicit, and incorporated in the process model hierarchical structure. It could be argued that a function's block type, and other pure data, should not be contained in the process structure since the structure itself is a rule network.

The rules regarding resource limits and class restrictions are handled "on the fly" by the engine. Therefore they cannot be said to be independently defined rules. Still this is a nice way of thinking of them so that is what we do. Otherwise they would have to be recognized as a part of the engine, but, since they actually are meta-rules, we can abide by the first definition. In a future version there would be a clear distinction between the engine and the meta-rules.

3.4.2 Data

The data operated on are a group of control stations and their content respectively as well as the names of the different parts of the process. Furthermore the dictionary provides the meaning of the names of function block types when these are needed.

3.4.3 Methodology

The methodology is also somewhat hazy in its definition. It could be regarded as a structured growth or optimistic incremental solution generation. Originally there was an effort to try to mimic the work of the expert. It is highly deterministic and provides a first solution which is then subject to scrutiny in order to find possible optimizations.

To start with there is an estimate of what a perfect system could ultimately consist of and this provides a "lower limit" for our target system. As data are collected from the process model the processes and functions found are continuously allocated to a control station. When a chunk of firmware does not fit, a new station is requested. Thus we get a first solution strongly connected to the implicit structure of the input process model.

The optimization is carried out by examining all resources present in all stations and then singling out the one used to the least extent. The functions dependent on the resource are determined (or selected heuristically when there is a set of candidates) and some other station with a sufficient amount of the same resource is located. A swap takes place and the process is repeated for the resource theat is now least used. The swapping continues until there is no possible swap or a system equal to the optimal, "lower limit", is found.

3.5 Implementational Issues

We will shortly touch upon some interesting details in the code. This is not a users manual and these comments are in no aspect a complete description of the implementation.

3.5.1 Data representation

Since the ZYX prolog is based on lists so is our data structures. We have throughout the system used n-ary trees based on lists using the following format:

(node_information (list of sons))

This format allows the use of general list traversing predicates such as map, member, split and so on. The alternative would have been to use abstract data types or some type of frames. However, demands on the representation has continuously changed during the development and lists representation was chosen as the most flexible. In a future revision, when data requirement will be clearer, abstract data types would probably be advisable.

3.5.2 Dialog System

The man machine interaction consists of both modal and modeless dialogs as well as menus. Since this is a main part of the tool we have developed a general dialog "manager". In short it works with Macintosh dialog resources and filters. Each dialog-box has its own filter that handles dialog events such as clicking a radio button. All data are handled by the filter and a correct data packet is returned only after a successful termination of the dialog, i.e. clicking the OK-button. Correctness test as well as consistency checks on data are also performed by the filter.

The modeless dialogs have been implemented using what ZYX call pseudo-processes. These are system "hooks" that are called when certain events occur, i.e. selection of a menu items or activating or typing in certain windows. The predicates called then make use of filters much in the same manner as described above.

3.5.3 Allocation Algorithm

As stated elsewhere this algorithm is highly deterministic. The input tree is traversed and the largest possible segments are allocated optimistically. The solution thus gained is then subject to optimization by attempting to swap functions to reduce superfluous resources. This is a variant of the travelling agency problem.

Unfortunately the system does not check for loops in the swapping chain. This could easily be remedied by providing a history trace much as is done in the blocks world problem to spot a previous state.

4 A Pulp Process, An Example Configuration

A modern pulp process is somewhat a typical example of a plant subject to computer control. It contain several independent parts with heavy machinery as well as pipelining and different monitoring systems. All these interact and may be integrated in a control system such as we have been describing.

4.1 Finding good examples

ABB has had a lot of experience in this field but unfortunately there is little on paper. The experts have been all too busy to provide us with the type of data that we would need to run thorough tests. Instead we had to make do with toy examples and some larger projects with rather incomplete information. The main issue here is that the computer system requires a lot more detailed information than does the human editor. A problem when there is no data and of course an asset when data is available.

4.2 A small, trivial, example

This is a so-called toy example, detailed in appendix A. It is not a model of a real production line but rather a small and stripped part of a plausible larger system. The function graph of the process contains only about 25 "granules" of data to be allocated. The resulting computation shows that the demands on the system from relations among sons become the ruling factor. The initial "optimal" system, prescribed solely by the requested relations, is not violated. More so, since the whole system could fit into one computer-station, the largest possible consistent units available does not cause any space violations. There is no need for considering what partitioning is the most suitable.

4.3 A large, not so good, example

Another example taken from a customer's specification of his system. The problem in this case, as in almost all "real" cases, is that all functions are listed in groups and almost all functional relations in between them have been removed. The effects of such simplifications are rigidness of structure and deterministic allocation due to the fact that nodes in the layer above the leaves become too large to be conveniently combined. The "optimal" configuration cannot be reached, but the number of process stations are determined by requested relations in this example as well.

5 Conclusions

We will not touch upon the effects of using a particular computer or environment for implementing the tool. Rather we would like to discuss the problems encountered and solutions suggested on the abstract level of modelling and expertizing. Included are also some suggestions for improvements and alternative approaches. Finally, particularly interesting avenues of further development are considered.

5.1 Modelling

There was already at an early stage of the development obvious reasons for using abstraction in the modelling of the three data structures needed. The problem was to identify relevant abstract data types. The solution stopped half way; we used associative lists with keyword access.

Also, the problem of placing data at the correct level has become cumbersome. The structure used has proven too rigid after considering the increased demands from the users. Nodes and leaves in the "demand" system should be regarded as being of the same type. If so, system demands could be allocated semi-globally for groups of functions, an example being, a local log printer.

Moreover the requirements on flexibility which concerns standard types could be met by using some object-oriented or at least hereditary structure. This would also mean improved understandability where access methods and relevant parameters are concerned. The abstract machine and the methods used to operate on data could be made more independent and hence a nicer modularization would be achieved.

5.2 Solution methods

The method used is sadly deterministic, only having a pathetic try at optimization. This is due to the fact that when the modelling of the process to be controlled is done, the allocation of functions to stations is rather closely guided. Apparent mistakes must then originate from the user defining the process model.

There is scant heuristic decision making and the cause for this is the lack of such knowledge. It would prove more useful to aid the designer in his modelling by providing typical solutions to situations common to the applications he is working with. For instance there could be a library of control structures covering all types of paper mill drying equipment.

5.3 Knowledge acquisition

Regrettably there has been little interaction with "live" experts at work. The system is mostly based on obvious information and descriptions of general configuration guidelines provided by ABB. System data has been readily available and there have been quite a lot of discussions concerning what level of detail would be preferable.

The system as it stands today could easily be filled up with "ground" data such as type functions and hardware performance. Since there is no real "expert knowledge" level in the system this prevents expansion of the knowledge built into the system. Instead there would have to be a revision of the system with the aim of entering an expert level of reasoning. Unfortunately, this expert reasoning has yet to be discovered.

6 Referendum

Bachant & McDermott: "R1 Revisited: Four Years in the Trenches", THE AI MAGAZINE, Fall 1984

Van de Brug, Bachant & McDermott: "The Taming of R1", IEEE Expert, 1986

Clancey & Rothenberg: "Evaluating Expert System Tools", AAAI87, 1987